

Introduction to Computers

Grades: 5



(Alan Turing, father of the theory of computability)

Objective: Students will simulate a computer processing an assembly script.

Time: 60 minutes

Materials:

- lot of post-it-notes
- pens/pencils

Overview:

Before modern day computers were invented, mechanical computers were used to run assembly scripts. Nobel Laureate Richard Feynman was head of the Los Alamos mechanical computer division in which he devised algorithms to perform complex calculations.

Today you will simulate a computer by first studying assembly scripts, simulating them, and then writing your own!

Scientific Background:

Students are simulating assembly instructions, the lowest level language aside from machine language. All computer programming languages that are compiled are produced in assembly format, which microprocessor companies such as Intel have standardized.

Students will be presented with a limited set of operations, and then together you and the students can create some scripts and run them. After that you may ask the students to write scripts that perform certain tasks and ask them to carry those out.

Computer Components:

Processor:

the processor carries out simple assembly instructions one after another.

RAM Memory:

RAM memory is temporary and depends on the electricity of the computer to support it. Memory is broken down into *registers* each of which can hold a numerical value. RAM access is extremely fast and is often used to store programs in use, etc.

ROM Memory:

ROM memory, such as harddrives, use magnetic tape or flash memory to create permanent storage. Memory is broken down into small aliquots sequentially (bytes).

BUS:

The BUS is a component that carries signals from different parts of the computer across the motherboard.

Display Device:

A method for the computer to display output to the user. This is usually the monitor and speakers.

Input Device:

A device used to enter input into the computer. This is generally the keyboard and the mouse.

Assembly Script:

MOV XYZ ABC

This statement copies the value of memory register XYZ to memory register ABC.

SET XYZ ABC

This statement sets the value of memory register XYZ to the value ABC.

ADD XYZ ABC

This statement adds the value of memory register XYZ to the value of memory register ABC.

SUB XYZ ABC

This statement subtracts the value of memory register XYZ to the value of memory register ABC.

READ XYZ ABC

This statement reads in the value of ROM location XYZ to the value of memory register ABC.

WRITE XYZ ABC

This statement writes the value of memory register XYZ to the value of ROM location ABC.

GOTO XYZ

This statement instructs the processor to continue reading in operations from memory register XYZ.

IF XYZ ABC

This statement does branch processing with the statement that if the value of memory register XYZ is not equal to 0, the processor is instructed to continue reading in operations from memory register XYZ.

Procedure:

1. Go through the above material on Computer Components and Assembly Script.
2. Setup the group as follows:
 - a. take 4-5 kids and let them stand together in a line. Each shall respectively represent one memory register.
 - b. Take one kid and let him be the processor. He/She should take in an instruction and direct the rest of the group (make sure they are only doing what they are allowed to as a processor).
 - c. Take 2 kids and let them represent the BUS, in that they can go and fetch information for the processor.
 - d. Have a group of 5-7 kids write a simple assembly script with your assistance. You should be comfortable in knowing how the program will be executed.
 - e. Have the group of 5-7 kids who wrote the script feed it into the processor one at a time. Information stored by the memory registers should be represented by a post-it-note where they write the value they have.

After the program has been processed, switch groups and repeat. Continue such to leave at least 10 minutes before end of lesson for discussion.

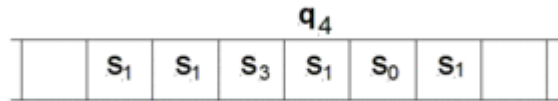
Additional Discussion/Questions:

1. Ask the kids how they felt about whether the other group correctly implemented their program.
2. Ask the kids how they felt about carrying out a program without knowing what was happening.
3. Discuss how much larger and faster actual computers are today compared to the kids. *1 GB RAM, 2 Ghz Processor, 100 GB harddrive.*
4. Discuss how the computer never tires and does not think yet can carry out complex sets of instructions if they are broken down.

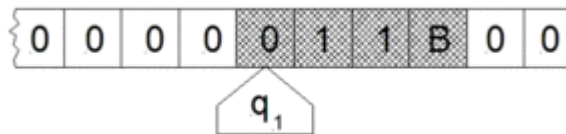
Turing Machine:

Wikipedia defines a turing machine as follows:

The concept of the Turing machine is based on the idea of a person executing a well-defined procedure by changing the contents of an unlimited paper tape, which is divided into squares that can contain one of a finite set of symbols. The person needs to remember one of a finite set of states and the procedure is formulated in very basic steps in the form of "If your state is 42 and the symbol you see is a '0' then replace this with a '1', move one symbol to the right, and assume state 17 as your new state."



In some models the tape moves and the unused tape is truly *blank*. The instruction to be performed (q_4) is shown over the scanned square. (Drawing after Kleene (1952) p.375).



In some models the *head* moves along the stationary tape. The instruction to be performed (q_1) is shown inside the head. In this model the "blank" tape is all 0's. The shaded squares, including the blank scanned by the head, and the squares marked 1, 1, B, and the head symbol, constitute the system state. (Drawing after Minsky (1967) p. 121).

More precisely, a Turing machine consists of:

1. A **TAPE** which is divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special *blank* symbol (here written as '0') and one or more other symbols. The tape is assumed to be arbitrarily extensible to the left and to the right, i.e., the Turing machine is always supplied with as much tape as it needs for its computation. Cells that have not been written to before are assumed to be filled with the blank symbol.
2. A **HEAD** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. In some models the head moves and the tape is stationary.
3. A **TABLE** of instructions ("action table", or *transition function*) that tells the machine what symbol to write, how to move the head ('L' for one step left, and 'R' for one step right) and what its new state will be, given the symbol it has just read on the tape and the state it is currently in. In some models, if there is no entry in the table for the current combination of symbol and state then the machine will halt; other models require all entries to be filled.
4. A **state register** that stores the state of the Turing table. The number of different states is always finite and there is one special *start state* with which the state register is initialized. Turing defined this as a "note of instructions" to preserve

the computation of the "computer" (a person) who is working in a "desultory manner":

Explain to the kids that their computer that they simulated was an implementation of a turing machine. Take questions on turing machines and emphasize that a turing machine can run any theoretical program possible.